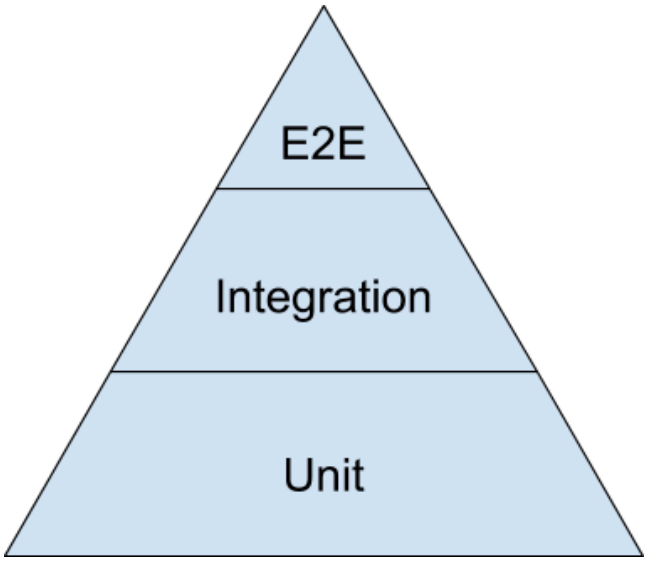
CS156 f25

lab / team03

team03 - Integration and E2E Testing

This team assignment is designed to introduce you to the ideas of integration tests and end-to-end tests, as illustrated in this diagram, often called the "testing pyramid":



This testing pyramid is explained further in the this video: team03 01 - Testing Pyramid (10 min) as well as in the following section.

Introduction to the Testing Pyramid

Briefly, the idea of the testing pyramid is that:

- Unit tests test only a *single unit*, so in principle, *absolutely everything* outside that unit should be mocked (including network services, database calls, etc.)
- For an integration test, you are testing the integration of multiple units, so it's ok to *not* mock some parts of the interaction. For example, an integration test of a controller might look *exactly like* the unit test of a controller, *except* that instead of mocking the database calls, you actually let the database calls happen to a real instance of the database (albeit one that is carefully controlled), and then you do assertions on the before and after state of the database tables.

- For an end-to-end test, you treat the entire application (to the extent possible) as a "black box", meaning that the tests typically have no knowledge of "what's inside the box". You use browser automation software to interact with an instance of a real browser (Firefox, Chrome, Edge, Safari, etc.) and to act like a "robot" that is clicking buttons and entering data on real web pages, and then making assertions about the content of the page that shows up.

You can read more about the testing pyramid in a dedicated article here:

- https://ucsb-cs156.github.io/topics/testing/testing_pyramid.html

If you were aware of the testing pyramid prior to taking this course, you might have noticed that unit tests exist in every assignment given in this course, but integration and end-to-end tests do not...yet. While the rolling in of the rest of the testing pyramid is in progress and while we decide how to best integrate these topics directly into the course material, this assignment is designed to be a brief introduction to these testing topics that you will likely encounter in real-world software development.

In this exercise, you'll do five things, working in the team repo from team02.

1. Run an existing integration test and look at the code.
2. Run an existing end-to-end test in headless mode (the normal way)
3. Run an existing end-to-end test in not headless mode (for development and debugging).
4. Develop an integration test of your own for *just one* of the controller methods from the controller you built during the team02 phase of the project that changes the database in some way (either create, update or delete; your choice).
5. Develop an end-to-end test for either create, update or delete for the user interface code you developed for a CRUD operations that changes the database (either create, update or delete, your choice).
6. Make a pull request containing these two tests.
7. Submit a link to the pull request on Canvas.

Part 1.1: Getting Started

(For a video covering this section, see: team03 02 - Integration Tests (24 min))

This team project will use the same repos and same Kanban boards that you used for team02. So you do not need a new repo or a new Kanban board.

Start by going into the directory where you cloned your team02 repo.

▶ Click triangle for links to the repos:

1 Begin by making a branch for this assignment.

2 Then, update that branch by merging in the code in https://github.com/ucsb-cs156-f25/STARTER-team02 again. You will need to do this in case there might be bug fixes to the code that supports integration and end-to-end testing that were added *after* most of your teams cloned the code for team02.

You may already have a remote for this called `starter`; you can check by typing: `git remote -v` and seeing if you have a remote called `starter`. If not, define one:

```
git remote add starter https://github.com/ucsb-cs156-f25/STARTER-team02
```

Then do:

```
git fetch starter
git merge starter/main
```

It is possible that there may be merge conflicts at this stage that you have to resolve; if so resolve those.

Then push to your new branch:

```
git push origin your-branch-name
```

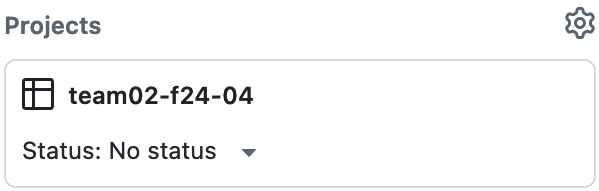
Now you are ready to try running the integration and end-to-end tests that are already in the starter code (instructions in next step.)

Part 1.2: Create two new issues

Still working in your team02 repo, you are going to create an issue for your work on this team project. *Each member of the team* should do this.

1. Go to your team's repo, and go to the `Issues` tab.
2. Click the **New issue** button at upper right.
3. For the title, use `Integration Tests for` *`DatabaseTable`*, replacing *`DatabaseTable`* with the name of the database table that you worked on in team02.
4. For the description, enter `Add at least one integration test for` *`DatabaseTable`*.
5. Save the issue by clicking the **Submit new issue** button.
6. Now we want to add this to the Kanban board. There are multiple ways of adding this to your Kanban board, but the easiest can be done directly from this page. Find the part of the page that looks like this in the right hand side:

Click on the gear. It will bring up a box where you can type in the name of your project for team02, e.g. `team02-f25-04`. Select this project. Then, the box should change to something like this:



Click the triangle beside `No Status` and change it to `To Do`. This will put the issue in the todo column on your Kanban board.

7. Do the same thing for a second issue, with title `End 2 End Tests for` *`DatabaseTable`*, and description `Add at least one end to end test for` *`DatabaseTable`*.

Part 1.3: Understanding integration and end-to-end tests

Before you begin working on the integration and end-to-end tests for the code you wrote in team02 and team02, you'll need to know how to run the tests that already exist in the codebase.

The starter code contains a number of Java test files, under the `integration` and `web` directories. These files use a couple of new tools that are explained in the testing pyramid article here. All of the integration and end-to-end tests contain the keyword `IT` which is similar to the keyword `Test`

that appears at the end of each unit test file, and we use `Web` to distinguish end-to-end test from integration tests.

- The `RestaurantIT` integration test class which contains the integration tests for the `Restaurants` backend. The structure of these tests are incredibly similar to the structure of the tests in `RestaurantControllerTests`, with the difference being that the integration test uses a real `RestaurantRepository` instead of a mocked one.
- The `HomePageWebIT` end-to-end test class that contains the minimum configuration necessary for running an end-to-end test within our stack. This test uses Playwright to assert whether or not the home page of our application contains the text that we expect.
- The `OauthWebIT` which tests our mocked authentication endpoint so that we can login like we normally do with Google Oauth but with fake user information.
- The `RestaurantWebIT` which contains the example of an end-to-end test for `Restaurants`.
- `WebTestCase` which is the parent class of `OauthWebIt` and `RestaurantWebIT` and contains much of the shared code and setup that is needed for the end-to-end tests.

These test files just contain a selection of examples of what kind of tests you could write for the backend and frontend that you worked on in team02 and team02. This assignment is relatively open-ended and any test that satisfies the submission criteria will be sufficient.

Part 1.4: Running the tests

You may be used to running `mvn test` in order to run the test suite for the application, but integration and end-to-end tests run with a seperate command.

In order to run the integration and end-to-end test suite you should use the following series of commands.

```
mvn clean
```

To make sure that we do not have anything lingering from previous test runs. Running `mvn clean` is important because the tests are highly sensitive and can fail if this is not done before the next steps.

```
INTEGRATION=true mvn test-compile
```

This step is this test compile command that has this `INTEGRATION=true` command at the front. What this does is it specifies that the program should run in the profile for integration tests. This command may take a while because it compiles the frontend with the backend.

```
INTEGRATION=true mvn failsafe:integration-test
```

This command actually runs the test suite. The test suite should pass.

If you have previously gone through these three commands and have **ONLY** modified the test cases then you may just use the last command, `INTEGRATION=true mvn failsafe:integration-test`. Otherwise, you may need to recompile.

If you want to just run a single integration/end-to-end test (e.g. only `HomePageWebIT.java`) use `-Dit.test=ClassName`, for example:

```
INTEGRATION=true mvn test-compile failsafe:integration-test -Dit.test=HomePageWebIT
```

A note about end-to-end tests: they can typically be done in two modes:

1. "headless", where there is no real browser being rendered on a screen; it's all just simulated in memory. This is the usual way of running end-to-end tests because its a lot faster.
2. "not headless" where the tester can actually watch all of the interactions happen on screen (albeit very quickly) as the tests are being run. This is typically only used when developing or debugging the end-to-end tests.

Our tests run "headless" by default and you can configure the tests to run "not headless" with the following:

```
INTEGRATION=true HEADLESS=false mvn failsafe:integration-test
```

Part 2: Your task

After understanding how to run the tests, you can begin working on the tests using the examples for the parts of the application that you work on in team01 and team02.

Part 2.1: Integration Test

Drag your issue for the Integration tests from `To Do` to `In Progress`.

Create a new branch from main (assuming that all PRs from team02 are merged into main) and begin working on the integration tests.

Create a file for your integration test with the name shown below.

Integration Test
`UCSBDiningCommonsMenuItemIT.java`
`UCSBOrganizationIT.java`
`RecommendationRequestIT.java`
`MenuItemReviewIT.java`
`HelpRequestIT.java`
`ArticleIT.java`

You can use the examples in `RestaurantIT.java` as a model, but it is not a requirement to follow it precisely and depending on your implementation you may not be able to. You just need a similar kind of test.

For the tests, you may choose any controller method you implemented in the previous part of the project to test or any frontend feature from team02.

You **are not** required to cover every controller method; we just want to be sure that you have experience with creating at least one integration test. (In a real app, we'd want to cover all of them, but the learning benefit of doing so starts to diminish after you've done the first one.)

When you've written at least one integration test and it passes:

1. Create a pull request for it
2. Drag the issue on the kanban board to In Review
3. Ask a team member for a code review
4. Move on to the end-to-end test

Part 2.2 End to End test

(For a video covering this section, see: team03 03 - End-to-end Tests (32 min))

Drag your issue for the Integration tests from `To Do` to `In Progress`.

Create a new branch from main (assuming that all PRs from team02 are merged into main) and begin working on the integration tests.

Create a file for your end-to-end test with the name shown below.

End-to-end Test
`UCSBDiningCommonsMenuItemWebIT.java`
`UCSBOrganizationWebIT.java`
`RecommendationRequestWebIT.java`
`MenuItemReviewWebIT.java`
`HelpRequestWebIT.java`
`ArticleWebIT.java`

For the end-to-end test you may choose any frontend feature related to the database you worked on in team02.

You can use the examples in `RestaurantWebIT.java` as a model, but it is not a requirement to follow it precisely and depending on your implementation you may not be able to. You just need a similar kind of test.

You **are not** required to cover every front end feature of the CRUD operations for your database table from team02; we just want to be sure that you have experience with creating at least one end to end test. (In a real app, we'd want to cover all of CRUD operations, but the learning benefit of doing so starts to diminish after you've done the first one.)

If your frontend form contains a date picker, you may have trouble selecting a date. If you can figure out how to do it with the Playwright Actions, great. If not, you can try to do something similar to the integration test, where you save something to the database using the repository directly.

When you've written at least one end to end test and it passes:

1. Create a pull request for it
2. Drag the issue on the kanban board to In Review
3. Ask a team member for a code review

Part 3: Submitting

As a team, every time class meets, do a standup and a review of the Kanban board.

As the deadline approaches, you may also need to:

- schedule time outside of class to check in with each other
- check in asynchronously at least once a day.

When all team members are finished, and all PRs are merged, submit a link to your repo at the Canvas assignment here: https://ucsb.instructure.com/courses/27687/assignments/381965

Videos

- team03 01 - Testing Pyramid (10 min)
- team03 02 - Integration Tests (24 min)
- team03 03 - End-to-end Tests (32 min)